Worked Examples and Exercises in OpenBUGS

These notes are to be used in conjunction with the workshop notes.

Contents

Fitting simple occupancy models with OpenBUGS.

Below is code that can be used to fit a simple occupancy model to the blue-ridge salamander data using OpenBUGS (an open-source version of WinBUGS; the line numbers are not part of the actual code). The code is in the OpenBUGS folder provided with the workshop materials. OpenBUGS is software that has been primarily developed for fitting models using the Bayesian approach to statistical inference, hence requires prior distributions to be defined for all model parameters, with the output in the form of (approximate) posterior distributions. Observations are specified as random variables from some defined probability distribution, and latent (unobserved) random variables can also be defined in a similar manner.

```
1:     model (psi(.)p(Day)) {
2:         for (ii in 1:39) {
3:             z[ii] ~ dbern(psi)
4:             z1[ii] <- z[ii]+1
5:             for (jj in 1:5) {
6:                 h[ii, jj] ~ dbern(p[z1[ii], jj])
7:             }
8:         }
9:         ### define prior distributions
10:        psi ~ dunif(0,1)
11:        for (jj in 1:5) {
12:            p[1, jj] <- 0
13:            p[2, jj] ~ dunif(0,1)
14:        }
15:    }
```

The OpenBUGS syntax requires that we define a model which includes both a definition of what probability distribution observations and latent random variables are drawn from, and the prior distribution for any parameters.

Our OpenBUGS code must therefore begin on line 1 with the statement ‘model {‘, and ends on line 15 with a corresponding bracket to close the model statement. Immediately following the word ‘model’ you can specify an optional model name in parentheses, as we have done so here.

The for loop on line 2 (and closes on line 8), is going to cycle the following code through each of the 39 transects for which we have data, where transects are indexed by the term ‘ii’.

Line 3 is where we define whether or not each site is occupied. z[ii] is the quantiy of interest, and in this case it is a binary valued or Bernoulli (0 or 1) latent random variable. It is latent in the sense that we cannot observe this directly because of imperfect detection. ‘~’ is used to denote that the quantity on its left-hand side is a random variable from the distribution on its right-hand side. And ‘dbern(psi)’ indicates that the distribution to be used is the Bernoulli with probability of a 1 (i.e., presence) equal to psi (i.e., the probability of occurrence).

There are a few different ways in which this same model could be fit within OpenBUGS, and this is of the form I currently favour; for an alternative approach see page XXXX in MacKenzie et al. (2006). Line 4 is simply there as a means to help with indexing. z1[ii] is a new quantity we’re defining, and that it’s equal to the value of z[ii] + 1 (‘<-‘ is the assignment operator in OpenBUGS and is equivalent to ‘=’). Therefore, if z[ii] is 0 (transect unoccupied) then z1[ii] is 1, and if z[ii] is 1 (transect occupied) then z1[ii] is 2.

Line 5 contains a second for loop, which closes on line 7, that will be used to cycle through each survey conducted at each transect. In this example there were 5 surveys of each site, and the index is jj.

The actual detection-nondetection data that was collected in the field is defined elsewhere in a 2-dimensional matrix ‘h’. In line 7 we are defining that the detection-nondection data is again a Bernoulli random variable, and the probability of detecting the species is p[z1[ii],jj]. Now recall that z1[ii] will either have the value of 1 or 2, so depending on whether the transect is estimated to be unoccupied or occupied, the probability of detection in that survey will be the jj^{th} value from either the first or second row of p respectively. Peeking down to lines 12 and 13, the probability of detection will therefore be 0 if the species is absent, or something else if the species is present.

Line 9 is simply a comment line to indicate that the following code is where the prior distributions for our parameters are defined. Anything on a line following a single ‘#’ is regarded as a comment.

On line 10 we are defining that our prior distribution for psi is a uniform distribution bounded by the values of 0 and 1. That is, it could be any value between 0 and 1, and all values are equally likely.

As we are assuming that detection probability is day-specific in this model, we must define a prior distribution for each of our 5 p’s, hence to for loop on line 11.

Line 12 is one of the important aspects of this code. The first row of p is the probability of detection given a transect is unoccupied. By definition this must therefore be 0 assuming no misidentification of the species. Then in line 13, if the transect is occupied, the prior distribution for p is again a uniform (0,1) distribution.

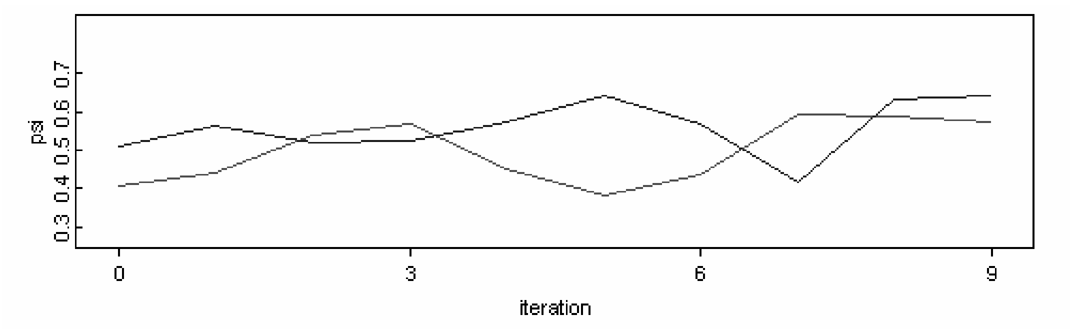
To run the model, double click on the word ‘model’ in line 1 so it becomes highlighted, then hit the **Check Model** button on the Specifications Tool window. In the bottom-left corner of the main OpenBUGS window a message should appear ‘model is syntactically correct’. If not, then carefully recheck your code.

Next we need to load the detection-nondection data. Within OpenBUGS, open the file ‘Salamander Data.odc’ which can be found within the OpenBUGS folder supplied in the workshop materials. This data is in a similar format to that used in PRESENCE with the data from each transect stored in a separate row, and the different surveys in separate columns. The first row in the data file defines which column of the data file corresponds to which vector in the data matrix. ‘h[,1]’ indicates that the corresponding column in the data file is the first column of the 2-d matrix h. The data file must be terminated with the statement ‘END’, followed by a hard-return. There are other formats that could be used to read data into OpenBUGS, check its documentation in the help menu for details. Once you have opened the file, make sure it is the active window then return to the **Specification Tool** window and hit **Load Data**. The message in the bottom-left corner should now change to ‘data loaded’.

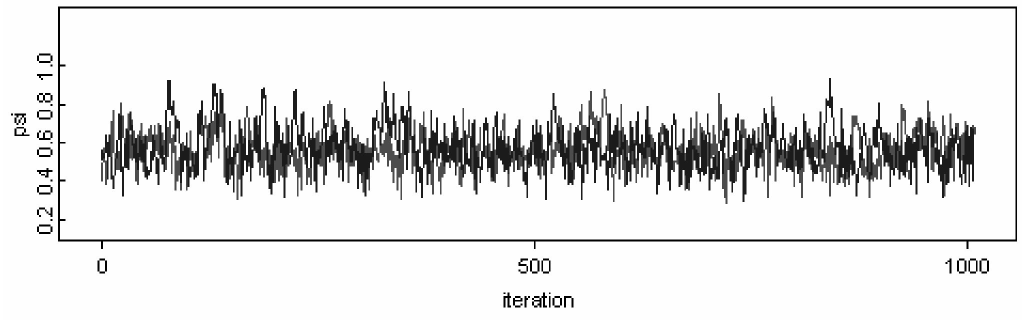
Following this, change the number of chains to 2, hit **Compile** then **Gen Inits**. The model should successfully compile then generate initial values for all unknowns. Here, the unknown values are the parameter values for psi and p, but also z[ii] as this is not observed data.

You must specify which unknowns OpenBUGS is to store for value for from each iteration of the MCMC procedure, i.e., from each sample from the posterior distribution. Open the **Sample Monitor Tool** window (found in the Inference menu) and specify that you which to store psi, p and z (hitting **set** after entering the name of each quantity).

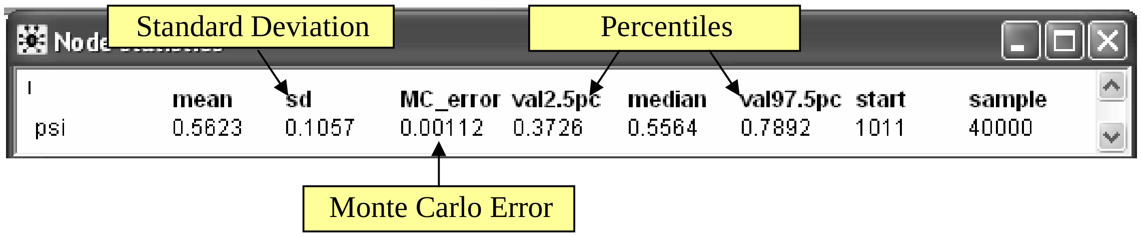
Finally open the **Update Tool** window (from the Models menu), specify to run 10 updates then hit the 'update' button. Return to the **Sample Monitor Tool** window, select 'psi' from the drop-down list box, and select the 'history' button. This should produce a plot similar to the one below. Each coloured line represents a chain of values for psi that have been stored by OpenBUGS.



Return to the **Update Tool** window and run a further 1,000 updates, then again examine the history for psi. Note that the 2 chains regularly cross one another and go through a similar range of values. This property is known as *mixing*, and is what you want to see in the output. If the chains do not mix, that indicates that the chains in the MCMC procedure have not *converged*. For more complicated models often it will take a number of iterations before the chains will have converged, and this information should be discarded (plus a few more iterations just to be safe) and is known as the *burn-in* period.

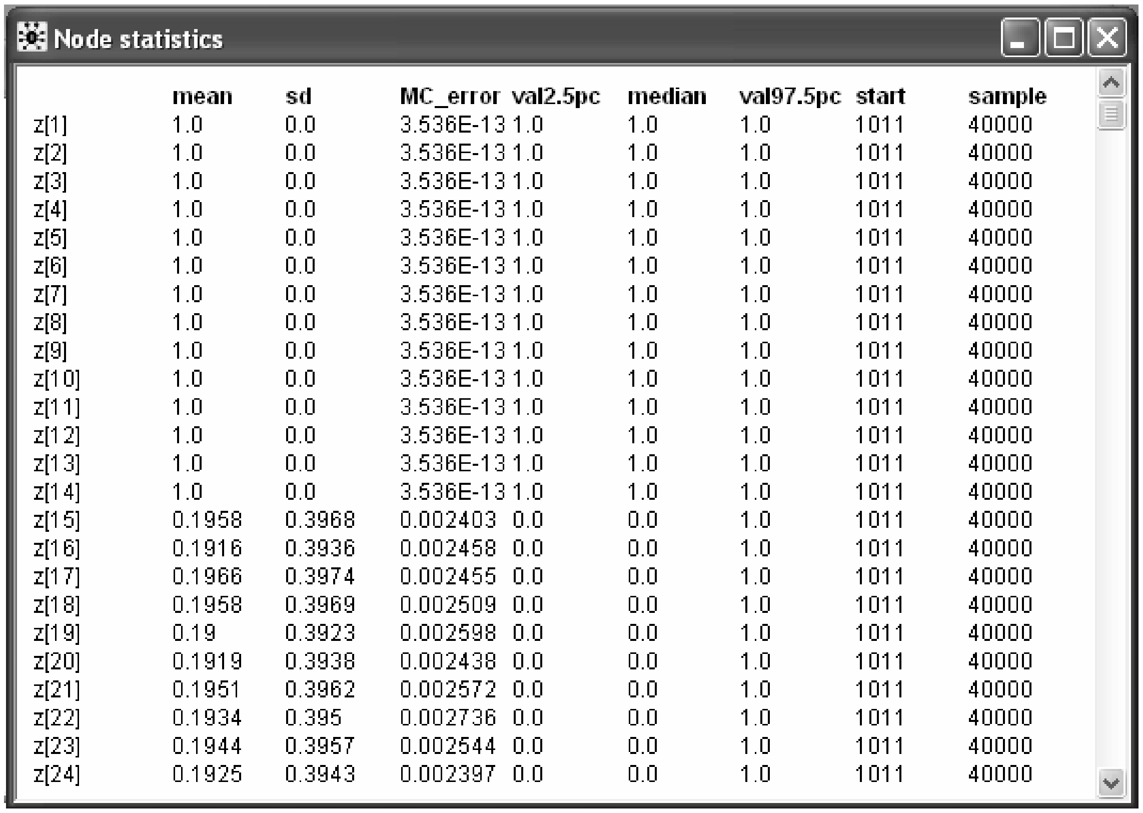


Run an additional 20,000 iterations and then we'll explore the output, discarding the first 1,010 iterations as the burn-in period. On the **Sample Monitor Tool**, change the number in the text box labelled 'beg' from 1 to 1,011 to set the burn-in period, then select psi and hit the 'stats' button. You should similar results to below being displayed, with most of the summaries for the (approximate) posterior distribution being self explanatory. One exception is likely to be the Monte Carlo Error, which is an estimate of the standard error to indicate how precisely the mean of the posterior distribution has been approximated given the number of iterations. It has been recommended this value should be less than 5% of the reported standard deviation. If its not then further iterations should be run. The percentiles can be used to form credible intervals (Bayesian equivalents of confidence intervals), e.g., here we could take the 2.5th and 97.5th percentiles to give a credible interval of (0.37, 0.79).



```
         mean    sd      MC_error  val2.5pc  median  val97.5pc  start  sample
psi      0.5623  0.1057  0.00112   0.3726    0.5564  0.7892     1011   40000
```

A plot of the posterior distribution can be generated by selecting ‘density’ from the **Sample Monitor Tool**. Next, explore the results for p noting that we do not get any for p[1,jj] as we’d specified these values to be fixed at zero. Finally, bring up the results for z, the unobserved variable associated with the presence and absence of salamanders (partially presented below). Essentially the mean of z takes on 2 values, 1 or ≈ 0.195, and is indicating the probability of occurrence given the particular detection history at each transect (and the model). Where the species was detected at least once, the mean of z will be 1 (it has to have been occupied), at all other transects salamanders where never detected, hence you’d expect the probability of occurrence given the nondetection to be lower than the estimates for psi.



```
         mean    sd      MC_error  val2.5pc  median  val97.5pc  start  sample
z[1]     1.0     0.0     3.536E-13 1.0       1.0     1.0        1011   40000
z[2]     1.0     0.0     3.536E-13 1.0       1.0     1.0        1011   40000
z[3]     1.0     0.0     3.536E-13 1.0       1.0     1.0        1011   40000
z[4]     1.0     0.0     3.536E-13 1.0       1.0     1.0        1011   40000
z[5]     1.0     0.0     3.536E-13 1.0       1.0     1.0        1011   40000
z[6]     1.0     0.0     3.536E-13 1.0       1.0     1.0        1011   40000
z[7]     1.0     0.0     3.536E-13 1.0       1.0     1.0        1011   40000
z[8]     1.0     0.0     3.536E-13 1.0       1.0     1.0        1011   40000
z[9]     1.0     0.0     3.536E-13 1.0       1.0     1.0        1011   40000
z[10]    1.0     0.0     3.536E-13 1.0       1.0     1.0        1011   40000
z[11]    1.0     0.0     3.536E-13 1.0       1.0     1.0        1011   40000
z[12]    1.0     0.0     3.536E-13 1.0       1.0     1.0        1011   40000
z[13]    1.0     0.0     3.536E-13 1.0       1.0     1.0        1011   40000
z[14]    1.0     0.0     3.536E-13 1.0       1.0     1.0        1011   40000
z[15]    0.1958  0.3968  0.002403  0.0       0.0     1.0        1011   40000
z[16]    0.1916  0.3936  0.002458  0.0       0.0     1.0        1011   40000
z[17]    0.1966  0.3974  0.002455  0.0       0.0     1.0        1011   40000
z[18]    0.1958  0.3969  0.002509  0.0       0.0     1.0        1011   40000
z[19]    0.19    0.3923  0.002598  0.0       0.0     1.0        1011   40000
z[20]    0.1919  0.3938  0.002438  0.0       0.0     1.0        1011   40000
z[21]    0.1951  0.3962  0.002572  0.0       0.0     1.0        1011   40000
z[22]    0.1934  0.395   0.002736  0.0       0.0     1.0        1011   40000
z[23]    0.1944  0.3957  0.002544  0.0       0.0     1.0        1011   40000
z[24]    0.1925  0.3943  0.002397  0.0       0.0     1.0        1011   40000
```

Exercise

Modify the code to fit models where; 1) *p* is the same for all surveys; and 2) *p* is the same for the first 2 surveys, and then constant for the last 3 surveys, but with a different value.

An Alternative Approach

There is an alternative method for defining the occupancy model in OpenBUGS that makes generalising the approach relatively straight forward. Rather than defining z as a Bernoulli random variable, we could define it as a categorical variable that can take 1 of 2 possible values (later we can generalise it to more than 2 values). Further, we need to redefine our observations as well. For simplicity, we'll just add 1 to everything so a 1 now represents absence and non-detection, and 2 indicates species presence and detection. Using this approach, the code for the salamander example becomes.

```
model (psi(.)p(Day)) {
    for (ii in 1:39) {
        z[ii] ~ dcat(phi[])
        for (jj in 1:5) {
            h[ii, jj] ~ dcat(p[z[ii], jj])
        }
    }
    ### define prior distributions
    psi ~ dunif(0,1)
    phi[1] <- 1-psi
    phi[2] <- psi
    for (jj in 1:5) {
        p[1, jj, 1] <- 1              # Pr(non-detection|absence)
        p[1, jj, 2] <- 0              # Pr(detection|absence)
        p[2, jj, 1] <- 1-p[2, jj, 2]  # Pr(non-detection|presence)
        p[2, jj, 2] ~ dunif(0,1)      # Pr(detection|presence)
    }
}
```

There is clearly more book-keeping required, but as we shall see later it makes generalising to other models much easier.

Defining a model with covariates

Including covariates into a model in OpenBUGS can be achieved, for example, with the following code using the weta data.

```
model (psi(Browsed)p(Obs)) {
    for (ii in 1:72) {
        z[ii] ~ dbern(psi[ii])
        z1[ii] <- z[ii]+1
        for (jj in 1:5) {
            h[ii, jj] ~ dbern(p[z1[ii],ii, jj])
        }
        ### define logit-link functions
        logit(psi[ii]) <- a[1] + a[2]*Browsed[ii]
```

```
            for (jj in 1:5) {
                p[1, ii, jj] <- 0
                logit(p[2, ii, jj]) <- b[1] + b[2]*Obs1[ii, jj] + b[3]*Obs2[ii, jj]
            }
        }
        #define priors for coefficients
        a[1] ~ dnorm(0, 0.07)
        a[2] ~ dnorm(0, 0.07)
        b[1] ~ dnorm(0, 0.07)
        b[2] ~ dnorm(0, 0.07)
        b[3] ~ dnorm(0, 0.07)
    }
```

The main things to note are that: 1) occupancy and detection probabilities are now indexed by unit (ii); 2) we make use of the built-in logit function to define the respective regression equations; and 3) prior distributions are now defined on the beta coefficients.